DOCUMENT RESUME

ED 428 582 FL 025 781

AUTHOR	Schmitt, Lothar M ; Christiansor. Kiel T.
TITLE	Implementation of a UNIX-Based Network Management System for English Instruction.
PUB DATE	1999-04-10
NOTE	49p.
PUB TYPE	Reports - Descriptive (141)
EDRS PRICE	EDRS Price MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS	Assignments; Classroom Techniques; College Instruction; *Computer Assisted Instruction; Computer Graphics; *Computer Networks; Data Collection; Editing; Electronic Mail; *English (Second Language); Error Correction; Foreign Countries; Higher Education; Homework; Programming; Second Language Instruction; Statistical Analysis; *Student Evaluation; Teaching Methods; Vocabulary; Writing (Composition); *Writing Instruction
IDENTIFIERS	Japan; University of Aizu (Japan); *UNIX Operating System

ABSTRACT

Pedagogical features and implementation of a UNIX-based management system (UNIEM) designed to support the instructor in teaching English as a second language using a network of workstations are described. The application discussed here is for teaching English composition to students at the University of Aizu (Japan). UNIEM is constructed to assist the computer novice in these tasks: sending out assignments using electronic mail (e-mail) on pre-set dates; reminders of missing homework when necessary; sorting of homework submitted using e-mail; enforcing submission deadlines and required homework length; partial evaluation of homework, particularly regarding mechanical mistakes such as spelling; distribution of evaluation results to students; reformatting of student writing for correction purposes; use of global or specialized vocabulary; identification and collection of authentic, interesting, or critical examples of grammar patterns for presentation in class; generation of statistical information; and graphic display of data. Implementation of these tools and potential for customization, possibly in connection with a specific research objective, are discussed. A program decomposition principle is also presented. (MSE)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED 428 582

Implementation of a UNIX-Based Network Management System for English Instruction

LOTHAR M. SCHMITT[1] AND KIEL T. CHRISTIANSON[2]

[1]The University of Aizu, School of Computer Science and Engineering,
Aizu-Wakamatsu City, Fukushima Prefecture, 965-80, Japan. E-mail: lothar@u-aizu.ac.jp
[2]Michigan State University, Dept. of Linguistics and Germanic, Slavic, Asian and African Languages,
East Lansing, Michigan 48824, USA. E-mail: chris118@pilot.msu.edu
[1,2] Both authors are equally main contributors to this publication. The authors are listed in non-alphabetical order due to evaluation procedures within The University of Aizu which do not recognize a main contributor properly unless listed as first author.

Abstract

This exposition discusses in detail implementation as well as pedagogical aspects of a UNIX-based management system (named UNEIM) which supports the instructor in teaching English as a second language using a network of workstations. The present implementation is aimed at teaching English composition to Japanese students at The University of Aizu.

UNEIM has a convenient setup mechanism which is designed to assist, in particular, the computer novice. The collection of functions that define the setup mechanism are described in detail to give a guideline for setup operations in similar teaching systems.

While running, UNEIM takes care of the following tasks using the `cron` *mechanism of UNIX: assignments are sent out via e-mail on preset dates; if necessary, students are reminded of missing homework; homework sent back by students via e-mail is sorted in regard to course, section and assignment; submission deadlines and required length of the homework are enforced; homework is partially evaluated in regard to mechanical mistakes such as spelling; results of the evaluations by the machine are sent back to the students automatically (to trigger resubmission); the writings of students are reformatted to make human correction easier; the use of global or specialized vocabulary can be measured for individual students as well as classes; authentic, interesting or critical examples of grammatical patterns can be identified and collected for presentation in class; desired statistical evidence is generated; and graphical display of data is generated. The implementation of these functions using UNIX tools is discussed as well as customizing the UNEIM system for a particular course possibly coupled with a specific research objective.*

We present a decomposition principle for programs (so-called applications). In fact, applications are decomposed canonically along the implemented hierarchically organized overall data structure which is usually traversed first top-to-bottom, and then bottom-to-top. This proved to be a powerful programming tool in regard to designing, extending and maintaining the UNEIM system.

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
This document has been reproduced as received from the person or organization originating it
Minor changes have been made to improve reproduction quality
Points of view or opinions stated in this document do not necessarily represent official OERI position or policy

PERMISSION TO REPRODUCE AND DISSEMINATE THIS MATERIAL IN MICROFICHE ONLY HAS BEEN GRANTED BY
Lothar Schmitt
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

BEST COPY AVAILABLE

FL 025 781

Foreword

The following exposition was first published (up to some changes) as Technical Report 97-2-008 of The University of Aizu. It discusses in detail technical and pedagogical aspects of a UNIX-based management system for English instruction named UNEIM. Later, the discussion of pedagogical aspects was extended and published together with a short survey of technical aspects in PEDAGOGICAL ASPECTS OF A UNIX-BASED NETWORK MANAGEMENT SYSTEM FOR ENGLISH INSTRUCTION, *System* **26** (1998), 567–589. By publishing the technical aspects of the system design with ERIC, the authors hope to give readers a guideline how to organize similar interactive or semi-interactive[1] systems. In particular, the organization of the administrative side of the UNEIM system is —as we think— easily portable, simple but very useful.

Introduction

The computer has the potential to relieve a great deal of the tedium associated with correcting the mechanical and basic grammar and vocabulary mistakes which are part and parcel of EFL writing instruction. By the same token, computers are suspiciously viewed by many EFL professionals as a threat to the autonomy, judgement, and value of the instructor. Typical of this suspicion is this recent statement by Denise E. Murray, past president of TESOL: "We have seen firsthand what can happen when technology is pushed on us by technocrats or 'techies' who don't ask for our opinions as the experts in language education..." [Murray 1996, p. 3].

Such sentiments (see also [Nunan&Lamb 1996, p. 197]), while often symptomatic of technophobia, are not unjustified. A survey of the literature and the software designed for language research or teaching shows that, essentially, there are two disjoint categories of systems: those designed for linguistic researchers and those designed for language teachers. The former are often extremely complex, which is necessary for their users whose research includes tasks such as morphological and syntactic analysis, compiling dictionaries, and writing descriptive grammars based on multi-million word corpora [Aijmer&Bengt 1991; Lee 1991; Souter&Atwell 1993], but are of little practical instructional value in the language classroom. The latter are

[1] Semi-interactive means here having automated e-mail communication, but not instant reply as with many internet applications.

often inflexible, expensive systems which necessitate that the instructor cede certain judgements of linguistic acceptability to the author(s) of the software [Chen 1996]. A prime example of these are the numerous grammar-checking programs such as Grammatik, PowerEdit, CorrectGrammar, and RightWriter, which are produced for native speakers and offer little or no assistance for non-native speakers [Murakawa 1995]. A great deal of software written specifically for EFL instructors is also limited in its flexibility by the fact that, except in the case of shareware, copyright laws prohibit the instructor from altering the software and distributing the altered version to students or fellow instructors and also by lengthy source codes written in standard programming languages such as C [Kernighan&Ritchie 1988] or Pascal [Jensen&Wirth 1991].

This paper describes the justification for design and implementation of a UNIX-based computer-assisted language instruction system using a network of workstations containing functions useful for instructors and students as well as researchers. From now on, the management system described in this paper will be referred to as UNEIM (UNIX network English instruction manager). Since the nature of UNIX is open, integrated, and free, UNEIM can be customized by any researcher or instructor. To alter or extend UNEIM, the user needs, for most cases, only to aquire some basic knowledge about the UNIX tools `sed` and `awk`. A short, complete, detailed introduction to `sed` and `awk` is given in [Christianson&Schmitt 1997]. This introduction is, in particular, customized to design applications and extensions of UNEIM. Many tasks needed by language researchers or instructors can be implemented with a few lines of code using the techniques described in [Christianson&Schmitt 1997].

While the CD-ROM TESOL/CELIA '96 lists about 70 freeware and shareware EFL programs and about 100 publishers' demos for the DOS/Windows operating system and a smaller number for the Macintosh operating system, to our knowledge, no other software exists which utilizes UNIX for EFL language instruction and research to the extent described below.

1 Developing a computer-assisted language instruction tool using UNIX

The University of Aizu is an international university with English as the working language. The overwhelming majority of students are native speakers of Japanese. All students are computer science majors and

obtain a mandatory education in English, which is the major language of communication and publication in their field of study. EFL instruction is provided by members of the Center for Language Research at The University of Aizu. Altogether, this has two main consequences in regard to our project: First, the Center for Language Research is fully integrated into the university network of UNIX-workstations. Every professor or student has a personal workstation available 24 hours a day. It is therefore logical for this network and UNIX to be utilized as a teaching device. This is different from the situation at other universities where there may be computers available for language research purposes but not for unlimited student and classroom use. Second, the students have basic computer skills. Thus, it does not impose an additional learning hurdle to use computers in English education. An interesting point for future discussion is whether or not it makes sense in today's technology-driven society to teach basic computer skills to non-computer-science majors in order use a system such as the one described here.

UNIX, the operating system of workstations in The University of Aizu, has several significant advantages over the Macintosh and DOS/Windows families of operating systems. Most importantly, UNIX provides a very rich variety of concepts, operations and tools which makes it easy to implement the system we have anticipated. Having the UNIX collection of tools readily available is very effective for the development of a prototype software system for assisting language instruction since many basic tasks, for example searching for strings of characters need not be reimplemented. The tools are fully integrated into the operating system. There are no obstacles such as different file formats or protocols, which may occur if individual components of a system are developed by different software makers. The tools gain additional power in combination with each other through the pipe mechanism of UNIX.

Important in regard to language analysis are for string searches the `grep`-family of filters [Kernighan&Pike 1984, p. 101], for most simple programming tasks `sed` [Kernighan&Pike 1984, p. 108; Christianson&Schmitt 1997] and `awk` [Aho et al. 1978; Kernighan&Pike 1984, p. 114; Christianson&Schmitt 1997], and for more complex tasks `lex` [Lesk&Schmidt 1978; Kernighan&Pike 1984, p. 256] and `yacc` [Johnson 1978; Kernighan&Pike 1984, p. 234] in connection with C and their counterparts developed by the free software foundation (GNU). In addition to the above, Mathematica [Wolfram 1991] is a more general programming language used to

generate graphical display of data.

`sed` is an ideal tool to alter, replace or simply identify strings in text. `sed` can be fully described in detail on a few pages and is easy to learn. Thus, UNEIM can easily be extented in regard to searching in text. Applications of searches conducted with `sed` include the collection of specialized vocabulary (in our case, vocabulary associated with computer science) and identification of interesting or critical grammatical patterns in text. We have shown how to set up such search mechanisms in [Christianson&Schmitt 1997]. `awk` is also a very good search tool, in particular, if numerical conditions such as a minimal number of words in a sentence are involved in the search. `awk` is, in contrast to `sed`, a full programming language which can be used to do numerical computations of all sorts. Using `awk` is very convenient. For example, variables in `awk` need no type declaration. `awk` can also be fully described in detail on a few pages and is easy to learn. Using `awk` allows alterations and extensions of UNEIM in regard to various methods of collecting numerical data with only a minor amount of work. Both, `sed` and `awk` are interpreters and do not require the user to compile code which otherwise may be an inconvenience.

UNEIM contains a collection of search mechanisms which are already implemented. In order to use these, the user has only to specify a list of words or phrases. UNEIM then generates the search programs to scan sentences for these words or phrases automatically. After being generated, these search programs are invoked by UNEIM without any additional action by the user. The results of the searches are collected for individual students and combined for courses. Another type of search that is already implemented is the measurement of specialized vocabulary used in student writings. Also in this case, the user has only to provide an additional word list in order to extend the present system. The generation and use of several types of associated programs is automatic: i) search programs for the words in the list, ii) programs that compute statistical data based on the search results, and iii) programs that display statistical data.

For the purpose of solving most tasks that are part of UNEIM, using `sed` and `awk` is sufficient. However, C possibly in connection with `lex` and `yacc` is more powerful and flexible, if really complex tasks such as specifications of large grammars have to be mastered. `lex` in connection with C can be used for detailed lexical analysis of text. `yacc` in connection with C allows the specification of grammars and uses `lex` most

of the time as lexical preprocessor/analyzer. See [Kernighan&Pike 1984, p. 233] and [Schreiner&Friedman 1985] for details of how to set up parsers and to analyze specific grammars using `lex` and `yacc` together with C.

Mathematica can be used for mathematical computations such as the differentiation of a function and for the display of statistical information through its graphic capabilities. Mathematica is not part of the UNIX system but is developed very much in its spirit so that it fits in nicely.

Besides the overall convenient design of the UNIX system, it is important to note that UNIX allows pseudo parallelism on a single workstation. Certain administrative tasks can be managed automatically by the operating system on the instructor's workstation without interfering with other work the instructor is doing with the machine. Examples are the sorting of homework from different courses into specific folders of the file system administered by UNEIM, the automated correction of certain types of mistakes in homework, and statistical evaluations. All such tasks can automatically be invoked by the `cron` mechanism of UNIX. Thus, the instructor usually only needs to look up the final results of evaluations and does not have to initiate them.

Finally, UNIX workstations in a computer network allow for true parallelism, since individual tasks can automatically be distributed to different workstations over the net.

2 UNEIM's structure

In this section, we shall describe in greater detail how UNEIM is designed. These design principles can, effectively, be used to implement other educational software systems.

2.1 Active side versus passive side

Roughly, UNEIM can be divided into a setup phase invoked by the instructor through a UNIX command `makeSetup` which is part of UNEIM and a running phase in which the procedures of UNEIM are initiated by `cron`. The separation into a passive developer's side and an active user's side avoids "open heart surgery" on the running system. Connected with the setup phase is the "passive" or static developer's side of UNEIM

which contains the source code for all main routines. One task of `makeSetup` is to copy or compile all routines from the passive developer's side to the active user's side. Connected with the running phase is the "active" or dynamic user's side of UNEIM which consists of code that is actually executed and the file system that is administered. The running phase uses procedures which are structured in accordance with the organizing principles described next.

2.2 The simplicity of getting started

During the setup phase which will be described below in more detail, the user has to provide only the following types of information:

The user has to set three strings of characters in a well-documented file (the header of `makeSetup`). These are `eSEMESTER`, `eTOPICS`, and `eTYPES`. `eSEMESTER` is a string of characters naming the current semester and is used to create several names of directories administered by UNEIM. `eTOPICS` tells UNEIM strings naming the different courses (topics) and sections (classes) the instructor teaches in a given semester. `eTYPES` tells UNEIM strings naming the different types of homework the instructor uses.

In addition to the above, the instructor has to provide one file per section which contains the login-names of the students in the section.

2.3 The active or dynamic structure

2.3.1 The three dimensions of a database administered by a family of programs

Essentially, UNEIM is developed around a philosophy of three dimensions of a database administered by a family of programs where the latter is considered as being part of the database. These are:

1. Time (e.g., year and semester, week in the semester)

2. Structure (e.g., the courses and sections for one semester)

3. Granulation or frequency (e.g., daily vs. weekly execution of programs)

The specific data in regard to the three dimensions can be combined to a graph as shown in Figure 1. This graph is modeled using the UNIX file system and associated UNIX file system commands. Note that a

Figure 1: Dimensions of UNEIM

non-hierarchical structure can be modeled using the UNIX command `ln` (link). In regard to our design, it is also important to note that we consider programs that access the data in the graph as data themselves and place them (at least virtually) in canonical places in the graph. This is indicated by the vertical line on top of the node `MoWe` in Figure 1. In fact, such a vertical line representing granulation is on top of every node in the horizontal Time-Structure plane. Programs that are placed "on top" of a particular node in the Time-Structure plane operate only on the data in that node and not on subdirectories and associated data. We require as a rule of design that every program which accesses, manipulates or generates data in the graph must be dissected into components that *can* be placed canonically. This will be discussed in greater detail below in the section explaining applications.

2.3.2 Modeling the three dimensions in the file tree administered by UNEIM

The dimension of time on a large scale is represented at the top layer of the file system administered by UNEIM. The dimension of structure is represented through the next lower levels of that file system. The structure that is to be administered changes in time in the large. In the case of language education, this means that the instructor teaches different or new courses (structure) every semester (time). Thus, every semester, a new file-tree for courses, sections and meetings is generated with `makeSetup`. As pointed out above, UNEIM provides a convenient mechanism to generate the full structure needed from a few entries in

BEST COPY AVAILABLE

`makeSetup`. For example, by setting

```
eSEMESTER='WS'
eTOPICS='Composition {MoWe TuTh} Grammar'
```

UNEIM generates the necessary structure for the winter semester of the current academic year and establishes folders and system files for two identical courses "Composition" (one presumed to meet Mondays and Wednesdays, the other Tuesdays and Thursdays) and one course "Grammar". We remark that the academic year changes by default April 5^{th}. This default can be reset in `makeSetup`. All strings such as `eTOPICS` are only used to create names of files and directories with no additional meaning. For example, instead of `eSEMESTER='WS'` one could set `eSEMESTER='ShoppingSeason'`. The second level where time is modeled on a smaller scale in the file system is the collection of folders associated with individual meetings (lessons) distributed over the semester. For example, UNEIM will generate the following folders if the above specifications are used:

- `Classes/96_WS_Composition/Assignment_01` up to `Classes/96_WS_Composition/Assignment_14`, `Classes/96_WS_Composition/MoWe_01` up to `Classes/96_WS_Composition/MoWe_14`, `Classes/96_WS_Composition/TuTh_01` up to `Classes/96_WS_Composition/TuTh_14`,
- `Classes/96_WS_Grammar/Assignment_01` up to `Classes/96_WS_Grammar/Assignment_14`, and `Classes/96_WS_Grammar/C_01` up to `Classes/96_WS_Grammar/C_14`.

The `C` in `C_01` is the default, which is used if no section, e.g., `MoWe` is specified. 14 is an upper bound which is sufficient in most cases but can also be changed conveniently in the header of `makeSetup`.

The third dimension "Granulation" is modeled as follows: UNEIM periodically performs certain tasks. For example, homework submitted by students via e-mail is collected every thirty minutes and evaluated every night, and statistics of things are computed every week. This is modeled using the file system in the following way (continuing the example given above): The folder `Classes/96_WS_Grammar` contains four subfolders

1. `Classes/96_WS_Grammar/S_30min`.

2. `Classes/96_WS_Grammar/S_Day`,

3. `Classes/96_WS_Grammar/S_Week`, and

4. `Classes/96_WS_Grammar/S_System`.

The first three folders contain shell scripts that are applied to the data (i.e., files and folders) in `Classes/96_WS_Grammar`. The scripts are applied periodically in accordance with the name of the host folder, i.e., scripts in `S_30min` are applied every thirty minutes. The fourth folder `S_System` contains files that describe the state of UNEIM and are seen as active all the time. They correspond to granulation 0. In a sense, every folder contains its own `bin`-folder [Kernighan&Pike 1984, p. 24], which is split into four parts, and the scripts residing therein "operate" on the data in the folder. Granulation folders of the type `S_30min`, `S_Day`, `S_Week`, and `S_System` are generated on all levels of the file system that is administered. The first three types are periodically applied to data in the corresponding folders in the file system. Thus, e.g., the `Classes` folder, as well as every folder `Classes/96_WS_Composition/MonWed_01` through `_14`, contains such granulation folders.

UNEIM is designed in such a way that despite the large number of local `bin`-folders there is, usually, only one executable copy of every program even though it may be executed on data in several folders on a particular level in the file hierarchy. How this is achieved will be explained below in greater detail.

The entire structure described above is generated through a single routine, namely `makeSetup`, which can be invoked repeatedly and does not overwrite existing structure. In this way, additional evaluations can be incorporated into UNEIM through the same procedure as the initial setup, which makes the user's interface very convenient. In addition, the setup is designed in such a way that some of the file structure administered by UNEIM is regenerated automatically to a virgin state in case it is lost.

2.4 The passive or static file structure

The overall structure of UNEIM is displayed in Figure 2. In addition to the file structure in the folder `Classes` which is generated by UNEIM, there are three main folders `Applications`, `Commands`, and `Files` that basically contain the source code for UNEIM:

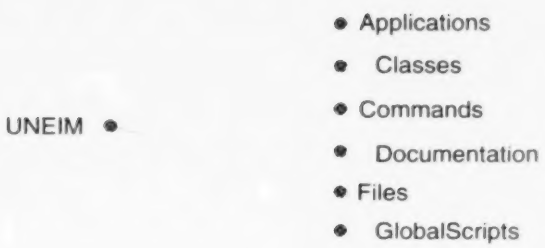

Figure 2: Structure of UNEIM

`Applications` contains sources of scripts and procedures which are executed on the data in the file hierarchy in the folder `Classes`. Essentially, every subfolder of `Applications` contains a family of related, interacting scripts that are copied into the granulation folders during setup. Such a family of interacting scripts is called an application. The folders in `Applications` are seen as part of the passive developer's side of UNEIM while the granulation folders are part of the active user's side. How a single application is organized will be discussed below in greater detail.

`Commands` contains a database of common scripts and programs including the setup operation. Most scripts and programs in `Commands` are used as subroutines in a number of applications described above. They are copied from the `Commands` folder into the user's `bin`-folder during the setup operation. Here, `Commands` is seen as the passive developer's side while the user's `bin`-folder is seen as the active user's side.

`Files` contains "data" such as headers of e-mail messages and default lists of words which are used in searches. Some files, e.g. headers of e-mail messages, are simply used and, thus, seen as active. Other files are seen as default, passive versions and are copied during setup to specific places in the file system administered by UNEIM.

2.4.1 Combining scripts to applications

Before giving a detailed description of the structure of programs in UNEIM (i.e., the structure of so-called applications), we state that, as general principle, execution of programs is organized similar to executing

"orders" or "executive decisions" in a hierarchically organized structure such as the military or a company. In fact, the hierarchy or file tree administered by UNEIM is first traverses *top-to-bottom* by a set of initial scripts, and then the hierarchy is traversed a second time *bottom-to-top* by a family of what could be understood as "report-type" scripts. The associated decomposition principle for programs (applications) proved to be a simple, yet powerful device which made designing, extending, and maintaining the system very convenient and efficient.

Suppose for a moment that the file structure given in the example above has been generated by UNEIM. As an example of a simple application, let us suppose that one wants to collect all sentences that contain the word "because" in all homework ever submitted in all courses that take place in the current semester. For the purpose of this example, this task can be over-dissected into the following 8 subtasks:

1. All-Courses-Together-Level (first): Clear the global collecting file containing all sentences with "because" from the last run of this procedure. With respect to our previous example, the global collecting file would reside in the directory `Classes` (possibly with a name indicating year and semester).

2. Course-Level (first): Clear the collecting file from the last run of this procedure containing all sentences with "because" for every course. With respect to our previous example, one of these collecting files would reside in the directory `Classes/96_WS_Composition`.

3. Meeting-Level (first): Clear the collecting file from the last run of this procedure containing all sentences with "because" for every meeting. With respect to our previous example, one of these collecting files would reside in the directory `Classes/96_WS_Composition/MoWe_01`.

4. Homework-File-Level (first): Reformat every single homework such that individual sentences are on separate lines. If the output of this procedure is needed for multiple evaluation, then save every reformatted homework in a separate, new file. This is one of the standard routines provided by UNEIM.

5. Homework-File-Level (last): Pipe or copy the output of 4) into a procedure that selects lines containing "because." This is, e.g., a one-line `grep`-script.

BEST COPY AVAILABLE

6. Meeting-Level (last): Append the output of 5) to the collecting file containing all sentences with "because" for the meeting the respective homework resides in which was emptied under 3).

7. Course-Level (last): Append all collecting files for the meetings pertaining to a specific course to the collecting file associated with that course which was emptied under 2).

8. All-Courses-Together-Level (last): Append all collecting files for courses to the global collecting file emptied under 1).

The program that collects all sentences containing "because" can be split into 8 `sh`-scripts each performing one of the subtasks just defined. Such a division of a program into several subscripts ignoring the loops through the file structure is called an application.

In general, an application is a subfolder of the `Applications` folder that contains a file describing its granulation and for each of the 8 levels just described zero or possibly several scripts. Looping through the file structure is designed to be performed by UNEIM and is ignored in the scripts. The names of the scripts are partly predefined. This is used to place the pieces of the application automatically in the granulation folders of type `S_30min`, `S_Day`, `S_Week`, and `S_System`. This approach has the following advantages:

- In the development of applications, parallelism becomes immediately evident. Different applications with the same granulation are obviously parallelisable. More importantly, distinct scripts on the same level are parallelisable. If so desired, they can be distributed automatically to different machines.

- Scripts for a particular level and granulation originating from different applications can be combined automatically to a single script if one does not want to distribute them to different machines.

- This approach supports modular development of programs, since the applications are naturally sliced into subtasks.

- The pieces of an application are logically together. Thus, editing by the developer is more convenient. To illustrate this point suppose the user has specified a punctuation check on a file containing homework in a script `homeworks.first.checkPUNCTUATION` in the folders containing the application that checks all homework. In order to implement an additional check, the above script can be used as a template

(i.e., copied and renamed) and altered appropriately specifying what has to be done with a single file containing homework in regard to the new check. That is all the developer has to do. The file specifying granulation in the folder containing the application determines the frequency of execution of the new check automatically. The new name which should start in `homeworks.first.` determines its place in the granulations folders and, thus, where (i.e., on which files) it is executed.

- The passive developer's side in the `Applications` folder and the active user's side in the granulation folders are separated.

Note that for the above simple task, only 1), 4), 5) and 6) are needed since the output of 5) can be appended directly to the global collecting file. However, the other collecting files may be useful if one wants to compare courses or meetings. Usually, an application does not contain scripts for all levels. For example, homework evaluation, which runs once a day, contains many parallelisable scripts on the lowest "Homework-File-Level," namely distinct checks and formatting operations. It does not contain anything on the "All-Courses-Together-Level." In contrast to that, the mail-handler contains a routine at the highest level which generates a temporary copy of the entire mailbox of the user. This assures the least interference between UNEIM and other programs that may alter the content of the mailbox such as a reshuffle or rename command of `exmh`. The mail handler does nothing on some lower levels since it recognizes homework and it's place by the identifier in the "Subject"-line of an e-mail.

Note: Actually, we have somewhat simplified the description of applications. In fact, there are "global" applications and "local" applications.

The scripts of the different global applications are placed together in a separate, global, `bin`-type folder named `GlobalScripts` which is not part of the file structure described so far. This folder is hierarchically organized exactly as the generated file structure in the folder `Classes` described above but does not contain directories for multiple courses, sections and meetings. In fact, it contains only granulation subfolders. Homework evaluation is an example of a global application. Thus, there is only one active, executable copy of each of the scripts that evaluate or reformat homework.

Local applications are supposed to be scripts that are applied to folders and files pertaining to specific

courses, sections or meetings. Hence, the instructor can customize the environment in regard to any need. For example, an exceptional type of homework may make a particular evaluation necessary only once per semester. Or one can search for specific target words depending upon each assignment.

The scripts of the global applications frame the scripts of the local applications. For example, on the lowest level the following order of execution is used: Global-Homework-File-Level (first) → Local-Homework-File-Level (first) → Local-Homework-File-Level (last) → Global-Homework-File-Level (last).

Note: Applications can be invoked by the user not only through placement in the granulation folders and subsequent execution through `cron`, but also directly. This mechanism is used in `makeSetup`.

2.4.2 The setup operation

The setup operation invoked by `makeSetup` is part of the `Commands` folder. It is a fairly large collection of routines which perform the following tasks:

- For security reasons, UNEIM is halted if already in use.
- A check on the format of arguments provided by the user is performed. The setup is interrupted if the format of the arguments is not correct.
- Out of the global arguments set by the user a file `shVar` with fixed data describing the state of UNEIM is generated. This file contains reference for many of the programs in `Commands` or, more precisely, their active copies in the user's `bin`-folder. In particular, `shVar` contains the definition of the file tree in the folder `Classes` which is administered by UNEIM. It is desirable to place the state of UNEIM into a separate file rather than to copy or compile the state into executable versions of programs. This retains maximal flexibility. For example, it allows the user to shift the running system temporarily to a different state, e.g., last year's classes. Such a procedure can be used to perform comparisons.
- The user's `crontab` file is first saved, then removed, and later reset appropriately. This procedure retains alien entries in `crontab`.
- A backup of the current state of UNEIM with all directories and files is generated. This part can be skipped by the user.

- Various commands are generated automatically. Examples are:
  - Out of a file that describes forbidden sequences of characters with regard to punctuation in matrix format, the punctuation check is synthesized.
  - Out of a list of abbreviations, a program is generated that identifies abbreviations in a given text file.
- The commands of UNEIM are copied to the user's `bin` folder. The latter is generated if it is not already in existence.
- The file system is generated. Making the file system is an application. (Essentially, all scripts contain a UNIX `mkdir` command.) The file system is generated with the *same* routine that is later used to operate on the file system and to maintain it.
- Various files are copied into the generated file system from default versions. For example, word lists are copied against which student writings are compared to monitor use of words related to computer science. Copying such word lists leaves the instructor the greatest amount of flexibility. For example, the generated word lists can be altered independently for every course depending upon the level of the course or progress made on the subject. On the other hand, an initial copy shows the novice user how to customize UNEIM for his or her special needs. Usually, the files that are copied contain detailed instructions for the user how they can be altered and what format must be observed. Another example of files that are installed at this stage are the flag-files that activate or deactivate folders for different kinds of processes. This part of the setup is also an application.
- The user is interactively guided to set up a student list for every section of every course. From that, a mailing list including the instructor is generated. Furthermore, the students are automatically notified that they are registered for the course. This part of the setup is also an application.
- The user is interactively guided to set up an alias for every course which students have to include into the "Subject"-lines of e-mail in order that UNEIM can identify homework properly. A suitable default

is generated for the user which he or she can change, if necessary. Usually, no action by the user is needed here. This part of the setup is also an application.

- Scripts of applications which are not part of the setup operation are placed automatically into global or local granulation folders of type `S_30min`, `S_Day`, `S_Week`, and `S_System`.

2.4.3 Other commands

The `Commands`-folder of UNEIM contains the data base of basic procedures which are combined to perform the different tasks of UNEIM. Besides the setup operation, these operations are:

- A global procedure which is activated in regard to granulation. Presently, this procedure is activated for the thirty minutes, one day, and one week granulation intervals by entries in the user's `crontab`-file. The latter file is generated or properly enlarged by `makeSetup` as described above. As a measure of security, executing the global procedure for the different granulations or the setup operation concurrently is not possible. In fact, these procedures block each other from execution via a flag-file. Thus, no script which is usually executed every thirty minutes is used while scripts are active that are invoked, e.g., only once a day. Overall, the user's `crontab`-file is arranged in such a way that the procedure pertaining to one-week granulation is guaranteed to be executed on Sunday nights, and the procedure pertaining to daily execution is guaranteed to be working all other nights. The global procedure initiates a loop over the file structure administered by UNEIM using the file `shVar` that contains the state of UNEIM which is generated by `makeSetup`. The global procedure is used to generate the file system as well as to operate on the latter via applications. It can be seen as "root" of UNEIM with exactly one "method" in the sense of object oriented design. In "specializations," only different scripts or tasks and different granulations are attached to it as outlined above in the description of applications. This approach makes it easy to change the overall structure and design of UNEIM, since, e.g., the definition of the file structure need only to be changed at one place. In addition, this is useful in regard to the modularity of UNEIM as a whole since every larger task or application in UNEIM is naturally dissected into smaller components. Furthermore, for every level of the file structure the global procedure demands a certain

format of the scripts attached. This forces the developer to adhere to a high degree of uniformity in writing scripts. Be reminded that there is also the possibility to invoke a specific application and the mechanism described here not only through `cron` and the granulation folders but also separately by the user at any moment in time.

- Certain checks for mechanical errors in text files such as occurrence of double words. These checks will be described in greater detail below in the context of user functions of UNEIM.

- Formatting operations. One example is the identification of sentences in essays submitted by students and reformatting the essays in such a way that whole sentences are on one line. This allows subsequently the scan for sentences that contain certain words, phrases or grammatical patterns which very often are indications of mistakes by students. How such scans can, conveniently, be initialized is described below. (For simplicity, we have assumed that an abbreviation never ends a sentence. Thus, sometimes UNEIM will generate double sentences on a line.) Another type of formatting operation is to rearrange the essays in such a way that suitable margins on both sides of the pages are maintained and the output is double spaced. This is used to print the homework that was submitted for human correction.

- Certain manipulations of files that mark, e.g., special cases of punctuation or abbreviations. Two examples are the programs `hideDIDNT` and `restoreDIDNT`. `hideDIDNT` marks strings such as `didn't` in text which use the apostrophe exceptionally in regard to punctuation. `restoreDIDNT` is the left inverse of `hideDIDNT`, if both are seen as maps acting on files.

- Set operations union, complement and intersection. These are useful if, for example, one wants to measure the occurrence of vocabulary. Sets are represented as lists of items (elements) with one item per line.

- Numerical operations. Files that contain numerical data pertaining to items such as words or phrases are formatted as one item per line with a number following the item, i.e., they are formatted as lists of type/token ratios. Such files can be seen as vectors over the free base of items [Greub 1981, p. 13]. In order to analyze the use of words and compute, e.g., accumulative word frequencies, certain vector

BEST COPY AVAILABLE

operations such as sum and difference are implemented.

2.4.4 Files of data

The directory `Files` contains a database of rather passive items. These include:

- A collection of headers of e-mail messages that are used by UNEIM. Most of these messages contain specific instructions for the student how to deal with and respond to a particular e-mail by UNEIM.
- Two lists of words that are searched for in text to measure special vocabulary. One is a list of special words related to computer science. The other is a list of "four letter words" and is intended to show how the user can extend UNEIM in regard to measurement of specialized vocabulary.
- Lists of certain patterns. These are used to synthesize search programs for sentences that contain these patterns. Identified authentic examples of student writings can be used by the instructor for presentation in class.
- A list of abbreviations, e.g., the string `bus`. This list is used to synthesize a program that identifies abbreviations in text.
- Certain pieces of programs that are synthesized by `makeSetup`.

2.5 Prototype development

2.5.1 UNEIM's history

The present state of UNEIM is that of a prototype software. UNEIM was first implemented as a collection of methods or programs that generate, access and manipulate data in a UNIX file tree as described by the graph in the horizontal Time-Structure plane in Figure 1. In the course of this development, a partial redesign was necessary and it turned out that developing programs outside the underlying graph results in a rather inflexible environment. This is due to the human tendency to write small programs that do a particular job and do not necessarily represent the whole structure of the underlying data-graph. As a consequence, the present design was implemented, which (at least virtually) integrates all programs in the same graph that contains the data these programs manipulate.

2.5.2 Advantages of the present approach

The emphasis of the present approach is to achieve the following goals:

- Generate as fast as possible a working environment. This uses the rich variety of tools available under UNIX rather than implementing everything in one specific programming language (preferably C). It is a positive feature of UNIX that for many tasks multiple solutions are possible using different tools. In the course of this project, it was necessary to implement the collection of functions useful for the students and the instructor/researcher altogether rather fast. To find useful functions was more important than to implement some in an optimal fashion. The fact that some code may be not optimal in regard to processing speed or usage of disk space is of minor importance at this stage: All procedures of UNEIM run sufficiently fast. The total disk space used for our project is less than 40MB, which is negligible in regard to costs of aquisition of hardware. Only after UNEIM is working as a prototype with all required functions, will we start to optimize the code in regard to processing speed and efficiency. Thus, at a later stage in the development process, most functions which have proven themselves to be useful building blocks for UNEIM will be implemented in faster versions which use as little disk space as possible.

- Generate a modular environment. By modularity, we mean that components of UNEIM can easily be altered with minimal interference to others. It turned out that the latter is particularly supported by our approach to dissect larger programs into applications. This process includes identifying crucial system tasks and dissecting them into reasonably small, reusable functions.

- Generate and use all functions of UNEIM from one source or root. For example, the routine that generates the file structure is also used to perform operations on the latter. As outlined above, only the scripts that are applied to the different layers of the file system are exchanged. Thus, the structure to be administered is fixed at one place and can easily be changed by changing one particular shell script.

2.5.3 Future implementations

In the future, a more efficient implementation of UNEIM is planned. One possible approach is to use classical database methods [Date 1994; Silberschatz et al. 1996]. Note however that such an approach makes it necessary to buy a database software system, and to buy or write software for implementation of a suitable user interface.

We have simply used files to record any result of data evaluation in UNEIM. This has the advantage that the results can be looked up by using the UNIX command `more` in the terminal. Another simple lookup method is to "click" at directories and files in an editor such as `emacs` or `mule`.

An alternative to this approach is to use HTML formatted files to display the results and to generate a suitable system of file links for an HTML browser which is separate from the underlying UNIX file tree. If this is implemented, then results could be generated only upon request through the browser which triggers execution of, e.g., a `perl` script [Gundavaram 1996; Wall&Schwarz 1990].

3 UNEIM's functions

UNEIM contains two collections of functions that are most relevant to the users, i.e., the students and the instructor/researcher. These are:

- functions that communicate with the students automatically, and
- diagnostic and administrative tools for the instructor/researcher.

Our philosophy is not to replace the instructor, and in this way, we empathize with concerns such as those expressed above by Denise E. Murray. Language is so human-bound and culture-bound that it is not reasonable to attempt to implement actual instruction on a machine. Our goal is to relieve the instructor from monotonous, repetitive duty and to assist him or her by providing a collection of diagnostic and administrative tools. In essence, UNEIM and its tools should provide more time in class for the human side of language teaching. Indeed, this is exactly the role of computers in the classroom that was observed in [Hardisty&Windeatt 1989, p. 90]: the teacher is "guiding, explaining, editing, keeping control, reassuring and encouraging, doing other class work, giving instructions, observing, and correcting;" the computer is

"memorizing, storing, providing a stimulus, moving and transferring information, saving time, anonymously editing, and printing."

3.1 Student functions

Altogether, the following tasks are performed at present by UNEIM in regard to automatically using e-mail to communicate with the student:

3.1.1 Registration

UNEIM notifies students that they are registered for the course during setup. In Japan, this is a useful service to the students indeed.

3.1.2 Confirmation of receipt of the e-mail

Homework is identified by the student's login name and a characterizing string in the "Subject"-line of the e-mail. UNEIM automatically makes a unique security copy of each submitted homework which is otherwise never processed and then either places the latest version of submitted homework into a corresponding folder, or rejects the homework if it is submitted too late or is insufficient in length. The student receives a copy of what has been stored under his or her ID. If the procedure overwrites an old version of homework, then that is mailed back to the student, too.

3.1.3 Error alerts

There are certain types of tasks in language teaching that can be automated without marginalizing the instructor. UNEIM automatically checks for the occurrence of the following mechanical mistakes:

Double words: In our own experience as well as in the experience of others [Kernighan&Pike 1984, p. 120], double words are easily generated through the use of, e.g., the copy-paste mechanism in text editors.

Punctuation and spacing mistakes: As outlined below, punctuation and spacing mistakes are particularly abundant in writings of Japanese students of English.

Spelling mistakes: The spell checker uses the UNIX command `spell`. We have compiled a "local" database of Japanese words and some technical terms such as HTML, which are communicated to `spell` to be correct. Consult the manual pages `man spell` to obtain a description how to include a specialized local database of words into `spell`.

All-uppercase-letter words: UNEIM identifies all-uppercase-letter words which are not part of the English language. This is intended to force the students to use language rather than all-uppercase-letter words as means to emphasize something in their writings.

Special letters: UNEIM identifies certain letters that are not part of written English, e.g., the underscore character or the equal sign. Again, this is intended to force the students to use language rather than fancy or funny writing styles to express something. The list of such letters can be explicitly set by the instructor in the file `badLetters` pertaining to every course. Continuing the example given above, one of these files is

```
Classes/96_WS_Composition/S_System/badLetters
```

A default version `.../badLetters` is generated automatically for every course. The instructor may edit each file `badLetters` in accordance with the instructions given in it to exclude or include letters. After having edited the file, invoking `makeSetup` will generate a search program and the associated error alert mechanism for the given list of letters.

The result of the analyses are mailed back to the students the day following submission. It is then the student's job to revise the homework assignment, correcting mistakes, and then to send the homework back to the instructor. We believe that these mandatory checks and the resultant succession of e-mail messages in the student's mailbox will prompt the student to proofread before mailing homework assignments to the instructor, thereby saving the instructor a great deal of busy work and teaching the student an important habit to get into.

We point out to the reader that in the Japanese culture the teacher is much more expected to guide the student than is the case in, e.g., Germany or the U.S. UNEIM provides a way of delivering a constant stimulus which students in Japan expect [Yang 1994].

BEST COPY AVAILABLE

Some of the checks listed above are also part of "The Writer's Workbench", see [McDonald et al. 1982] and [Kernighan&Pike 1984, p. 120]. We have not used any code of that software package. However, our search for double words is based upon a program in [Kernighan&Pike 1984, p. 121].

3.2 Instructor functions

3.2.1 Sending out assignments

The setup operation generates well documented flag files `keepDue.`*type* for every type of homework in every folder `S_System` pertaining to a meeting. This flag file steers the mailing of assignments. In regard to the example listed above pertaining to the directory `Classes/96_WS_Composition/MoWe_01`, one of the flag file generated would be

`Classes/96_WS_Composition/MoWe_01/S_System/keepDue.ICW`

if `ICW` (in-class writing) were one of the homework types. If the file is edited by the instructor in accordance with the instructions given in that file, then UNEIM will mail the file `assignment.ICW` in the directory

`Classes/96_WS_Composition/Assignments_01`

to every student in the section that meets Mondays and Wednesdays. In fact, the instructor has only to put the word `due` in the first line of `keepDue.ICW`. In addition to the above, all students that have not yet submitted some homework in regard to the assignment will be notified of that fact every Tuesday and Friday morning that follow the initial mail. In that case, the file `assignment.ICW` will be mailed to those students once more.

Again, we point out that this mechanism provides part of a continuous stimulus to students which is expected to come from teachers in the Japanese culture more so than in western educational systems [Yang 1994].

3.2.2 Size enforcement

Students are notified, and their work is rejected, if the length of their work is considered to be insufficient, i.e., contains less than 100 different words. The limit 100 can be set to another value in the header of `makeSetup`.

3.2.3 Printing homework I

UNEIM delivers the first version of every homework that is submitted in reformatted style to the instructor. The mail header is removed while the student's name and other relevant information is retained. In addition, the homework is suitably formatted for human correction; the output is double spaced and suitable page margins are used. The instructor can choose between having the reformatted homework printed over night on a specific printer whose name is to be set in the header of `makeSetup`, or (by default) having the collected homeworks sent to him by UNEIM in a single e-mail which he or she then can print after arriving at the office. Printing the first version assures immediate feedback to the student.

3.2.4 Collecting homework

UNEIM collects all versions of a particular homework assignment of a particular student in a separate file. This is a simple tool for the instructor when determining the student's progression and to compile writing portfolios.

3.2.5 Deadline enforcement

The instructor can close folders associated with meetings for accepting homework. If we continue the example given above, then pertaining to the directory `Classes/96_WS_Composition` the instructor has to edit the file

`Classes/96_WS_Composition/MoWe_01/S_System/openClosed.ICW`

to discontinue acceptance of homework in regard to `assignment.ICW` in the directory

`Classes/96_WS_Composition/Assignments_01`.

If a student attempts to submit homework after the first line of the flag file `openClosed.ICW` contains the word `closed`, then UNEIM will send the homework back to the student with an explanatory note saying that the deadline has passed.

3.2.6 Printing homework II

One day after a folder pertaining to a meeting is closed via editing the associated flag file, UNEIM will collect and print all latest versions of all homeworks pertaining to the meeting. Again, the homework is suitably

formatted for human correction and grading. Printing the final versions together allows comparison. Also, it shows all improvements due to instruction and automated correction.

3.2.7 Specifying word lists

Currently, UNEIM has two categories of words that are specially looked for. First, UNEIM searches for words related to computer science [Orr et al. 1995]. Measuring the increase in use of vocabulary related to computer science is interesting to us since the students at The University are computer science majors. This system lends itself to the instruction of any sort of specialized vocabulary associated with various fields of study. Second, we have included a collection of "four letter words" as a demonstration of how to set up the search for other word categories whose decrease in use one may be interested in measuring. In fact, to alter both collections of words, the instructor has only to edit the files `essentialWords.computer` or `essentialWords.fourLetter` in the `S_System` folder pertaining to a course (or the default file in the directory `Files`). To continue our example given above, one folder to look for those two files would be

```
Classes/96_WS_Composition/S_System
```

The two files are generated from default versions in the folder `Files` and can be customized locally for every course.

If the instructor wants to include a third search category of words, then he or she has to do only the following:

1. Alter the list of search categories in the `sh`-variable `eBUZZWORDS` in the header of `makeSetup` in accordance with the description given in that file. For example, the user could set

   ```
   eBUZZWORDS='computer fourLetter japanese'
   ```

   extending the current setting by a search list containing Japanese words. It would be interesting to measure the use of Japanese words in essays by Japanese students compared to overall use of vocabulary over time as an indication of the (hopefully) increasing autonomy of their English mental lexicon from their Japanese mental lexicons, in accordance with the parasitic strategy model [Hall 1992].

2. As the second and final step, the user would have to generate a file

`Files/essentialWords.japanese.default`

which would be copied during setup as usable default version into the `S_System` folders pertaining to every course. As above, the default versions could then be altered "locally."

UNEIM is constructed in such a way that collecting statistics as well as generating graphics in regard to the third word list would also be immediately triggered by the above two steps without further action by the user.

3.2.8 Selecting example sentences

As outlined above, one function of UNEIM is a rather simple but efficient mechanism to identify sentences. This is used to collect example sentences that contain potential mistakes which are typical for beginning students due to erroneous one-to-one translation between, e.g., Japanese and English. A collection of typical mistakes of Japanese students is given in [Webb 1992]. A list of words or phrases the instructor wants to search for can be set in the file `scanWords` pertaining to every course and every meeting. Continuing the example given above one of these files is

`Classes/96_WS_Composition/S_System/scanWords`

another could be

`Classes/96_WS_Composition/MoWe_01/S_System/scanWords`

A default version of `scanWords` is generated automatically for every course. It is not generated for every meeting. A file `scanWords` pertaining to a course may be edited in accordance with the instructions given in it. Also, the instructor may generate a file `scanWords` of the same format in the `S_System` folder pertaining to a meeting. After having edited or generated a file `scanWords`, invoking `makeSetup` will generate a search program and the associated selection mechanism for the given list words or phrases. One can also specify in `scanWords` that if a particular pattern is found, then a given number of preceeding sentences is printed. This can, e.g., be used to check for pronoun reference or the logical connections between sentences (i.e., cause-effect) as expressed by conjuncts such as "as a result."

UNEIM handles multi-word strings in the following way:

- If the instructor wishes to locate all occurrences of "By the way" or "by the way" with no intervening

words, he or she simply includes the string `by_the_way` in one of the files `scanWords`.

- If, on the other hand, the instructor wishes to search for "put off," a separable phrasal verb, he or she simply includes the string `put off` in one of the files `scanWords`, and UNEIM will locate and print all sentences containing the string `put` followed by the string `off`, whether they occur together or separated by one or more words, as in "put the meeting off."

3.2.9 Keeping statistics of things

UNEIM keeps statistics on a number of items. Examples are:

- The number of words used in an essay.
- The number of unique words (words appearing only once).
- The type/token ratio of every word in an essay.
- The number of words in the special search categories used in an essay.
- The number of sentences used in an essay.
- The number of paragraphs used in an essay.
- The length of sentences (in words).
- The length of paragraphs (in sentences).

Statistical information is collected in additional files which are automatically generated. The data pertaining to individual students are also combined in order to obtain data for entire courses that show, e.g., the use and increase of specialized vocabulary in the course.

3.2.10 Displaying data

We have used Mathematica to implement the display of numerical data. In fact, every sort of numerical data collected in files is displayed as a two or three dimensional plot showing either meeting vs. numerical data for courses or (meeting, student) vs. numerical data.

4 Pedagogical justifications and research applications

4.1 Providing additional stimulus

In Japan, the role of the instructor is often different from the role of the instructor in western educational systems [Yang 1994]. One duty which is expected much more is to guide the students towards teaching goals rather than to force the student to train on his or her own by means of strict selection and passing mechanisms. However, such an understanding of teaching may be a significant burden for the non-Japanese instructor due to cultural barriers and due to the sheer number of students in the class.

UNEIM provides one way to implement a constant dialogue by machine for the instructor with his or her students. Reminders on due homework provide a somewhat unpleasant stimulus to work. The automated correction of mistakes is a reminder to maintain a certain level of quality. This guides the student constantly without taking its toll on the instructor who may therefore save energy for more interesting and humanly rewarding aspects of his or her duties.

4.2 Punctuation and spelling

Especially at the beginning of a school term, an EFL instructor spends a great deal of time correcting mistakes in the mechanics of composition, reminding students of the importance of proofreading, and training students to get into the habit of doing so. Prominent examples of mistakes in the mechanics of composition are punctuation and spacing as well as spelling. Correcting such mistakes becomes an even more time-consuming process when the EFL students are native speakers of a language with a radically different orthography with no inter-word spacing and a limited set of punctuation marks such as Japanese. In fact, the Japanese language (outside of 'manga' or comic books) has only analogues of the period, comma, and quotation marks and no spacing between words or after punctuation [Saito 1992]. Furthermore, many students in today's computer-centered educational institutions are asked to write on the computer despite having had little or no practice in keyboarding in even their native language. The result is that an EFL instructor spends a significant portion of his or her time marking spelling and punctuation mistakes in student writings.

Punctuation, in terms of spacing and ordering (not usage), were described with a finite set of forbidden

two-sequences of characters which can easily be checked by machine. UNEIM identifies non-standard strings containing punctuation marks (e.g., a blank or a tab before a period) in each submitted homework assignment, then sends the assignment back to the student, informing the student that there are possible mistakes and marking those possible mistakes for correction. The student should then check the possible mistakes, correct them as necessary, and resubmit the homework assignment.

After one semester of pilot teaching with UNEIM, students' punctuation and spacing have improved quite dramatically. In fact, mistakes in regard to punctuation and spacing were practically gone after the first two weeks of the semester while in previous semesters they constantly occurred in large numbers over the whole term. This strongly suggests that the automated, obstinate correction by an *unintelligent* machine is a much more stringent reminder mechanism than correction by humans. Furthermore, the specific nature of mistakes is much more separated and, thus, emphasized which, in most situations, is impossible to achieve for the human instructor due to lack of time. This helps students to focus and, consequently, to improve their quality of work. As found by [Ferris 1995], students respond more favorably and more productively to teacher comments on initial rather than final drafts. This kind of on-going response from the earliest stages is what UNEIM has been built to facilitate.

In order to provide the student with a spell-checker, UNEIM has been designed to check all submitted homework assignments with `spell`, the spell-check tool available under UNIX. Students whose homework contains possible spelling mistakes are notified in the same way as for punctuation mistakes. This mechanism of UNEIM forces students to actually use `spell` and to carefully proofread *before* submitting their work. Again, the machine is an obstinate companion in the course who never loses the energy to "fight" on this low level of teaching English composition.

One problem with `spell` is that a great number of specialized and foreign words, along with abbreviations and acronyms are not listed in its dictionary. This problem, however, can be easily solved by adding these lexical items to `spell` in a separate file. A list of hundreds of common and uncommon abbreviations and acronyms (especially those relating to computer science, e.g., HTML, NeXT, LaTeX, even DOS and IBM) has been installed in UNEIM. Also, a list of hundreds of Japan-specific lexical items have been installed,

including first and last names, famous people from Japanese history and pop culture, food, and city and prefecture names.

It should be noted again that there are software packages available which are designed for some of the tasks described here, such as the Writer's Workbench [McDonald et al. 1982; Reid 1990], but few of these are adaptable to classroom FL purposes such as document management, automatic statistics generation, and example sentence generation. Moreover, none of them comes with an e-mail and file management system like UNEIM. From a general point of view, these alien tools are more or less programs similar to `spell`.

As indicated above, automated processing of mechanical types of mistakes makes correction of submitted text in regard to real content much easier since there is no need to focus on the most trivial levels.

4.3 Monitoring language development

Lewis states that language consists of grammaticalized lexis, not lexicalized grammar [Lewis 1993]. If this is true, as a great deal of subsequent research dealing with Lewis's lexical approach suggests, then UNEIM is ideally designed, despite certain crudities that have been retained in order to ensure ease of use, not only for monitoring vocabulary use and development as measured by the frequency of correct usage, but also for monitoring total language development. Since lexical items are the "minimal units for certain syntactic purposes" [Lewis 1993, p. 90], performing a string search for certain lexical items and printing out the sentence(s) in which they occur allows the instructor to generate sample sentences and lessons which highlight not only individual vocabulary items, but also basic grammatical constructions. Moreover, this can be done without having to write or use an enormously complex and time-consuming grammatical tagging system (e.g., [Leech&Fligelstone 1992; Meyer&Tenney 1993]).

Lewis also stresses the importance of collocations, multi-word items, and polywords, what are often referred to as phrasal verbs or idioms, such as "put off", "the day after tomorrow", "by the way", and "look up to." Lewis considers items such as these to be individual lexical units, and claims that knowing how these units fit together, i.e., collocational knowledge, constitutes the foundation of linguistic knowledge, and he finds support from Willis who writes, "If one starts by listing words and their behavior, one can generate automatically the structural environments and the words which are likely to occur within them" [Willis 1990,

BEST COPY AVAILABLE

p. 13].

UNEIM provides a built-in way to encourage and monitor the use of a specific set of lexical items. Suppose the instructor has defined a list of specific lexical items that have been judged to be essential for students to learn at a given level. The instructor can introduce these items to students, and students can be told to use a specific number of the items in the next writing assignment. UNEIM can search for the special lexical items in the instructor's list, tally statistics on their use, and print out example sentences containing them. From any given point onward, then, UNEIM can record statistics both by student to monitor individual progress, and by lexical item to monitor frequency of use. The former helps in determining who might need some remedial instruction or further practice, and the latter helps in determining the relative difficulty of each lexical item. Example sentences, both correct and incorrect, can be used to direct students' attention to collocational and usage information for the lexical items.

It is important to note that recent research suggests that collocational knowledge is acquired in a certain order [Gitsaki 1996]. Activities such as the one described here can be graded to expose students first to the types of collocations which are acquired earlier and those which are more common.

4.3.1 Monitoring vocabulary use

In general, there are two ways to measure increases in a foreign language student's active vocabulary: (1) in terms of quantity, or (2) in terms of content. Both, (1) and (2) can be measured using the processes implemented in UNEIM. (1) is basically a rough measure of the student's relative fluency, i.e., "How many words is the student writing per assignment?" Such a measure would be particularly informative when the variable of time is constant, as in timed writings [Caudry 1990; Henry 1996]. (2) can to a certain extent be measured by the number of new words a student uses in each successive assignment, as compared to all of the student's past assignments, i.e., "Is the student trying to use new words, and if so, which words and in which order?" We have implemented this using the set operations union and complement. We remark that the statistics and word lists generated by applying these measures to student writings would be valuable in many aspects of lexical acquisition research.

4.3.2 Searching for sentences containing specific words or phrases

A component of many EFL courses is some sort of directed vocabulary or lexical study, including collocations, idioms, and multi-word items. However, simple memorization of lexical items is not enough; the items must be used in production to complete the acquisition process [Gass&Selinker 1994, p. 283].

As outlined above, one of the instructor functions of UNEIM is to automatically generate a file of sentences taken from student writings which contain a given list of words or phrases (`scanWords`). To use our previous example, UNEIM can find all sentences containing the string "because" and print all such sentences in one file. Furthermore, since "because" is a conjunct, the appropriateness of which often depends on the previous sentence, UNEIM is set up in such a way that it will also print the sentence immediately preceding the one containing "because" along with information identifying the student, the number of the meeting and the section.

A string search can be done on both the global level, i.e., on all homework all semester for the most virulent of reoccurring errors, and on the local level, with the instructor adding a list `scanWords` of more specialized words to search for to the `S_System`-folder pertaining to a particular meeting. For Japanese students, "because" would probably be searched for on the global level, as it is a constant source of confusion [Izzo 1995].

To illustrate another typical usage of the search mechanism for sentences in UNEIM, let us consider usage of "while" and "during." The instructor might, after reading a number of assignments containing problems with these words, decide to search the cumulative body of student writings up to a given point for all occurrences of these words, and then use the results of this search to introduce the topic in class and allow the students to develop their own rules. Furthermore, if the instructor has noticed the tendency on the part of students to use "in" where "during" or "while" would be more appropriate (a common mistake among Japanese EFL learners), "in" could also be added to the search. In that case, the instructor would, obviously, need to read through the generated example sentences to filter out occurrences of "in" that were used in a different sense. Such a global search (implemented as an application as in the example searching for "because" given above) can be invoked directly at any point in time aside from the use of granulation

folders.

Collocations as well as non-idiomatic collocations can be located by simply telling UNEIM to look for one of the key words in the collocation. For example, Japanese EFL learners often use the word "play" in connection with activities that do no co-occur with it, such as judo (i.e., "play judo"), skiing, fishing, karate, and roller-blading [Webb 1992, p. 41]. By searching for `play`, `plays`, `playing`, and `played`, the instructor can locate all sentences in the student writings containing non-idiomatic collocations such as the above along with idiomatic ones such as "play poker."

Such a tool, while simple, is extremely useful to the language instructor. Any lexical and numerous syntactic problems can be isolated in student writings, collected into one file, printed out, and then used in classroom exercises with students. From a classroom management standpoint, automating this process saves the instructor hours of retyping or cutting and pasting. The instructor need not even read the student writings to locate problematical words or sentence constructions. A quick proofreading of the collected file to discard sentences that for one reason or another are not illustrative of a given grammar point is enough.

From a pedagogical standpoint, using the students' own writings from which to draw examples is much more attractive than using "canned" sentences from grammar books or which have been made up by the instructor. In the words of Omaggio, "students are more motivated when language-learning materials seem relevant to their communicative needs and interests and resemble authentic language use" [Omaggio 1986, p. 313]. At the same time, by stripping the students' names and ID numbers from the sample sentence file, the students' errors remain anonymous, their correction in class by other students unthreatening.

We remark that in future extensions of UNEIM the files containing example sentences could be "cleaned" from information identifying students, and then could automatically be sent to all students in a class for additional exercise like identifying inappropriate uses of a given pattern.

4.3.3 Locating grammatical patterns

Some larger patterns, such as verb tenses, can also be identified with more or less accuracy using simple string searches. For example, to locate sentences in the simple past tense, UNEIM need only to search for strings:

BEST COPY AVAILABLE

- contained in a list of the past tense forms of irregular verbs, or
- containing six or more letters and ending with `ed`.

Likewise, for passive constructions, UNEIM would search for sentences containing `am`, `is`, `are`, `been`, `was`, `were` and in those search for strings

- contained in a list of past participles of irregular verbs, or
- containing six or more letters ending and with `ed`.

The list needed for the first type of search can be found in many comprehensive English grammars, such as [Quirk, et al. 1985].

The accuracy of this approach, however, requires that the grammar construction in question has been well formed, which in much EFL learner writings it is not. A cruder approach, then, would be to search for and print all sentences containing "have" or "has" to locate occurrences of the present and past perfect tenses, thereby avoiding missing malformed perfective verb forms. Crude as such an approach is, analysis of the current level of sophistication of grammar checking software suggests that they "lack the knowledge needed to analyze sentences and text structures in an appropriate way" [Jansen 1994, p. 239].

As discussed above, certain string searches tend to yield examples of certain syntactic constructions. Therefore, even without a sophisticated tagging scheme, UNEIM provides a starting point for the grammatical analysis of a text. This string-search based approach can be used to analyze grammatical constructions in a lexico-semantic framework, such as that described in [Dixon 1991]. Briefly, Dixon proposes an English grammar based on semantic principles. His approach is predicated upon a semantic classification of English "content" words (i.e., nouns, adjectives and verbs), and analysis of the syntactic, collocational, and functional restrictions the words in these classes impose on sentence construction, e.g., their behavior when suffixed with "-ing" or when complimented with "to" or "for." Text analysis within this framework can easily be carried out with UNEIM by conducting string searches for the specific lexical items in each semantic class. This can be used to compare native speakers' and non-native speakers' use of grammar.

Note: We show in [Christianson&Schmitt 1997] how to set up tagging programs and how they can be used in connection with pattern matching to perform grammatical analysis, the uses of which being very much in the spirit of [Kerim-Zade&Pavlov 1989] and [Owen 1993].

4.4 Concordancing

In [Kennedy 1991], the collocational properties of "through" and "between" are analyzed by counting and classifying all the words that directly preceded and directly followed both words. Kennedy also collected example sentences containing all "tokens" of the two words. He used the Oxford Concordance Program [Hockey&Martin 1988] to perform his search. In [Renouf&Sinclair 1991], the collocational structure of phrases such as `a .* of` and `many .* of` is investigated.

A researcher can use the techniques described in [Christianson&Schmitt 1997] to analyze any size of corpus in a similar or the same fashion. Such searches can be included into UNEIM, e.g., as research applications comparing native speakers' and non-native speakers' use of certain lexical items and analyzing the use of those lexical items in relation to time or content of lectures. The major difference in using the techniques described in [Christianson&Schmitt 1997] would be that instead of the corpus being tagged for word classes and grammatical features (as is the case with all large research corpora), any tagging would be built into the search programs itself. As such, if such a tagging system were to be installed in UNEIM, the user could search any corpus of text, again saving the expense of purchasing access to commercial corpora, such as the Brown Corpus, the London-Lund Corpus, or the Gothenburg Corpus of Collocations (see [Barlow 1997]).

This being said, the main point to be made here is that the typical teacher/researcher need not purchase package software to begin concordancing and exploring either large corpora or student corpora or, most important for a language teacher, comparing native and learner corpora.

Note: Using the techniques described in [Christianson&Schmitt 1997], one can search through documents on the World Wide Web for occurrences of certain lexical items and grammatical constructions. The resulting language corpus can then be used for concordancing activities in which the students compare the occurrences of apparently synonymous lexical items for differences in their collocational characteristics or to check the idiomaticity of their writings with the corpus. This kind of student-directed and/or instructor-lead linguistic

BEST COPY AVAILABLE

inquiry is proposed in [Tribble&Jones 1990] and [Scott&Murison-Bowie 1993].

4.5 Statistics and display thereof

As mentioned above, the EFL and/or second language acquisition researcher has access to a wide range of statistics and their display which analyze writings inputed into UNEIM. Such statistical information can be used to measure e.g., improvement in the use of the language in terms of fluency and productivity in timed and untimed writings.

In the present version of UNEIM, student homework is sorted according to two types: In-class writing (ICW) and out-of-class writing (OCW). This dichotomous categorization was used mainly because of the fact that out-of-class writing, while not significantly different from timed in-class writing tasks [Kroll 1990b], presents a different set of problems for students in regard to time spent planning the writing and the total amount of actual output (for example, the total number of words or sentences, and sentence length). Furthermore, in a recent survey of American college professors who teach content-area subjects to international students, a full one-third of the survey respondents stated that writing under time pressure in English "was a skill foreign students should develop" [Kehe&Kehe 1996, p. 113]. The statistical metrics that UNEIM applies to student writing would provide an excellent tool for research into the relationship between students' production, time variables, environmental variables, and even topic-related variables.

Most of these statistics are similar to or the same as the ones found in packaged commercial software such as Grammatik or the Writer's Workbench. There are, however, some differences. First of all, UNEIM can check a piece of writing against wordlists compiled by the instructor, i.e., customized to the needs of the students. On the other hand, UNEIM does not have, as of yet, a way to judge the overall level of the writing based on the statistics generated.

Statistics allow the instructor or researcher to measure the lexical density of a piece of writing. Lexical density is defined as "the ratio of content carrying words to total running words" [Lewis 1993, p. 100]. Measuring lexical density gives the instructor or researcher a more accurate picture of the level of the writing. Generating a measure of lexical density is relatively simple. Non-content words in English tend to be those words classified as "closed-class" or "minor" word classes, e.g., pronouns, prepositions, determiners,

conjunctions, auxiliary verbs, and interjections. These classes cannot easily be added to. Content words, conversely, are referred to as "open-class" or "major" word classes, e.g., nouns, verbs, adjectives, and adverbs [Leech&Svartvik 1994]. UNEIM already contains lists of the "closed-class" words, against which it can check and then count all the words in a given piece of writing not matching these lists.

Mathematica is used to generate customized display of graphical data. A surprising result which became immediately apparent from one of the displays generated by data collected during the first semester of UNEIM testing at The University of Aizu is that every student seemingly has a constant, *personal* rate by which the vocabulary used in his or her essays increases. In other words, every student uses an almost constant number of additional words in every new essay.

4.6 Maintaining writing portfolios

Since UNEIM automatically saves old versions of student writings, it is possible to maintain extensive portfolios of student writings, measuring student progress both quantitatively with statistics and qualitatively with automatically generated files and printouts of students' collected writings. Teaching FL writing using a portfolio approach is gaining popularity, largely because of its wide-spread use in teaching native-speaker composition. It is described by Leki as "based on assembling a representative sample of the student's best work...[and] at the end of the term, the entire portfolio is evaluated for a grade" [Leki 1990, p. 65]. It is immediately apparent that the record-keeping, document-management difficulties that such an approach would entail could be easily eliminated with UNEIM.

5 Additional applications

The following sections outline applications of UNEIM that have yet to be implemented or which are only partially implemented. None of them require any major revisions of or additions to the existing UNEIM architecture.

5.1 Learner-specific trouble spots

Any EFL instructor is immediately able to enumerate dozens of errors typically made by students of various native-language backgrounds. Any instructor is aware of the inordinate amount of time that is spent correcting and commenting on these "typical" errors. UNEIM allows an instructor to search specifically for such errors and generate e-mail automatically to assist the students in evaluating and, if necessary, correcting these errors.

For Japanese EFL learners, [Webb 1992] identifies 121 common mistakes. Over 75 of these can be located with simple string searches and the sentences in which they occur can be automatically printed into a file and used for in-class activities. More importantly, a great many of these mistakes are so-called "air-tight" mistakes, the location of which by UNEIM initiates a return in an e-mail to the student containing the (possible) mistake, and explanation of the problem, and a suggestion for how to correct it. For example, Japanese students often write "By the way" [Webb 1992, pp. 159-160, mistake No. 118] in formal writing to either connect related information to previous information or to introduce a main point, neither of which is acceptable in standard English writing. As such, when UNEIM encounters the string "by the way" (capitalized or uncapitalized) it would send back the following e-mail:

```
Possible mistake: >By the way<, the castle is the most important sight in the city.
Explanation:  "By the way" is used only to introduce information that is not directly
              related to the other information in your writing.  It is used mostly in
              informal writing such as letters or e-mail to friends or in speaking.
              Therefore, avoid using "by the way" in formal writing.
Alternatives: Perhaps you can use "Also," "In addition," "Additionally," or
              "Incidentally" in place of "By the way."
```

5.2 Automatic vocabulary practice

Vocabulary testing can also be done automatically by UNEIM. From a file of 1000 prominent words, (compiled by the instructor based upon the vocabulary needs of each individual course), each student receives

weekly an individual test. A list of 1000 words with unique or a dominant translation out of the 2000 most common words is compiled in [DeHart&Schmitt 1997]. All tests taken together present the 1000 words in random order. By testing 36 words a week for 28 weeks (the average length of a school year in Japan), each student will see all 1000 over the course of the year.

5.3 Automated tests

Tests can be designed such that they can be automatically evaluated and corrected by UNEIM. The tests can be designed in such a way that every student receives an individual task. Within the context of a typical Japanese university, where language classes meet only one time a week, this sort of automatic testing serves

- as a review of the material covered in the previous class period; and
- as motivation for students to use and study the language outside of the classroom.

5.4 Evaluation and Grading

UNEIM can be used to perform all sorts of accounting in regard to final or intermediate grades for the students. Such grades can also be communicated to the students via UNEIM.

5.5 Report generation

UNEIM can be used to combine statistics, evaluations and other data about a course into a comprehensive report.

Conclusion

The present UNIX-based computer-assisted language instruction system is unique in that it employs the considerable resources of a the UNIX programming environment for the purposes of foreign language instruction. Furthermore, UNEIM is flexible and can be quite easily altered to fit the needs of the individual language instructor, something which, usually, cannot be done with packaged commercial software.

The main features of UNEIM are:

- UNEIM is embedded in the university network which allows communication via computer between students and instructor.
- Pseudo parallelism is used to deligate repetitive tasks away from the instructor to the machine without having the need to initialize this work on the machine and with the ability to use the machine for other tasks at the same time.
- Preexisting tools of UNIX are employed to generate an open, flexible, high level environment of procedures in a very short time which can easily be altered or extended.
- UNEIM contains routines that administer homework submission, partially correct mechanical type of mistakes and send the corrections back to students, and analyze student writings in regard to use of vocabulary, selected lexical items and grammar.
- Statistical data generated by UNEIM is displayed graphically.

UNEIM has been pilot-tested in one first-semester composition course at the University of Aizu, and plans are being made to test it in the composition and technical writing courses during the coming semesters. The results of the application of UNEIM during the initial pilot-testing are:

Motivation for students

UNEIM was seen positively by students. Dealing in part with a computer as instructor raised their curiosity as computer science students. It is not clear what effect the use of UNEIM had on class attendance, but the attendance throughout the semester was nearly perfect, much better than that of most composition classes in the previous semesters.

Similarly, all but two students out of 26 consistently submitted revised versions of homework, corrected according to the automatically-generated e-mail the students received which alerted them to possible spelling and punctuation mistakes.

Homework from all of the students, especially out-of-class writing (OCW) grew in length from assignment to assignment, and the persistent formatting (i.e., paragraphing) problems which had been observed over six

BEST COPY AVAILABLE

previous semesters of composition instruction (when most writing was done primarily by hand) disappeared within the first month of class.

The possibility to resubmit homework after it was corrected by the machine was a reason for students to do additional work. The 24 students in the course submitted a total of 386 drafts of writing assignments, 126 revised drafts beyond the 260 completed final versions. This represents an average of nearly 5 extra drafts per student over the course of the semester. More interestingly, the number of submitted revisions dropped as the semester progressed because students began to proofread their writing before submitting it to avoid receiving mistake alerts via e-mail from UNEIM.

As such, the daily feedback on trivial mistakes had a drastic impact on the quality of what students delivered. Punctuation and spacing mistakes were nearly eliminated.

Motivation for the instructor

The instructor was able to concentrate class time and written feedback more on the less tedious and more crucial aspects of teaching EFL writing, namely dealing with global issues such as text cohesion, thesis development, and organization.

Also the interaction via computer with the students provided an additional means of communication which actually made teaching easier.

Administrative tasks like sorting, filing or generating (formatting) printouts were done by the machine automatically, saving the instructor approximately one hour a day during the semester. When the time normally spent on correcting spelling and punctuation-spacing mistakes is taken into account, as well as the time spent collecting either handwritten assignments or hardcopy assignments and organizing/inputing them into the computer for research purposes, is tallied, the time savings were enormous. As such, as highlighted above, teaching can be combined with research in a more efficient way.

References

Aho, A. V., Kernighan, B. W. & Weinberger, P. J. (1978). awk – A pattern scanning and processing

language (2nd edition). In: B. W. Kernighan & M. D. McIlroy (Eds.), *UNIX programmer's manual (7th edition)*. Murray Hill, NJ: Bell Laboratories.

On-line `troff` file: `http://cm.bell-labs.com/7thEdMan/vol2/awk`

Aijmer, K. & Altenberg, B. (Eds.) (1991). *English corpus linguistics*. New York: Longman Publishers.

Barlow, M. (1997). *Corpus linguistics page*. Department of Linguistics, Rice University, Houston, Texas, USA. `http://www.ruf.rice.edu/~barlow/corpus.html`

Caudry, T. (1990). The validity of timed essay tests in the assessment of writing skills. *ELT Journal*, 44(2), 122-131.

Chen, J. F. (1996). CALL is not a hammer and not every teaching problem is a nail! (Changing expectations of computers in the classroom). *The Internet TESL Journal*, 2(7).

`http://www.aitech.ac.jp/~iteslj/Articles/Chen-CALL.html`

Christianson, K. T. & Schmitt, L. M. (1997). Combining the Bourne-shell, sed and awk in the UNIX environment for language analysis. Preprint. The University of Aizu, Aizu-Wakamatsu, Japan. Submitted.

Date, C. J. (1995). *An introduction to database systems*. Reading, MA: Addison Wesley Publishing Company.

DeHart, J. A. & Schmitt, L. M. (1997). Preprint in preparation. The University of Aizu, Aizu-Wakamatsu, Japan.

Dixon, R. M. W. (1991). *A new approach to English grammar on semantic principles*. Oxford: Oxford University Press.

Gass, S. M. & Selinker, L. (1994). *Second language acquisition: An introductory course*. Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

FERRIS, D. R. (1995). Student reactions to teacher response in multiple-draft composition classes. *TESOL Quarterly*, 29(1), 33-50.

GITSAKI, C. (1996). The development of ESL collocational knowledge. Unpublished doctoral thesis. Center for Language Teaching and Research, The University of Queensland, Brisbane, Australia. `http://www.cltr.uq.oz.au:8000/users/christina.gitsaki/thesis/contents.html`

GREUB, W. (1981). *Linear Algebra*. Graduate Texts in Mathematics 23. Berlin: Springer-Verlag.

GUNDAVARAM, S. (1996). *CGI programming on the World Wide Web*. Sebastopol, CA: O'Reilly & Associates.

HALL, C. J. (1992). Making the right connections: vocabulary learning and the mental lexicon. Research Report (143). Universidad de las Americas-Puebla, Puebla, Mexico.
ERIC Document Reproduction Service (EDRS), Doc. No. ED363128. `http://edrs.com/`

HARDISTY, D. & WINDEATT, S. (1989). *CALL*. Oxford: Oxford University Press.

HENRY, K. (1996). Early L2 writing development: A study of autobiographical essays by university-level students of Russian. *Modern Language Journal*, 80(3), 309-326.

HOCKEY, S. & MARTIN, J. (1988). *The Oxford concordance program: User's manual (Version 2)*. Oxford: Oxford University Computing Service.

HUME, A. G. & MCILROY, M. D. (EDS.) (1990). *UNIX programmer's manual (10th edition)*. Murray Hill, NJ: Bell Laboratories.

IZZO, J. (1995). Usage of subordinating conjunctives by freshman students. IN: J. Izzo (Ed.), *1994 Annual Review of The Center for Language Research*, 37-42. The University of Aizu, Aizu-Wakamatsu, Japan.
ERIC Document Reproduction Service (EDRS), Doc. No. ED394284. `http://edrs.com/`
`http://www.u-aizu.ac.jp/~izzo/Subconj94/subconj94.html`

JANSEN, C. (1994). Computerized writing aids: do they really help? IN: M. Stechouder, C. Jansen, P. van

BEST COPY AVAILABLE

der Poort & R. Verheigen (Eds.), *Quality of technical documentation*, 239-248. Amsterdam: Editions Rodopi, B. V.

Jensen, K. & Wirth, N. (1991). *Pascal user manual and report (ISO Pascal standard)*. Berlin: Springer-Verlag.

Johnson, S. C. (1978). Yacc: Yet another compiler-compiler. In: B. W. Kernighan & M. D. McIlroy (Eds.), *UNIX programmer's manual (7th edition)*. Murray Hill, NJ: Bell Laboratories.

On-line `troff` file: `http://cm.bell-labs.com/7thEdMan/vol2/yacc.bun`

Karim-Zade, I. & Pavlov, V. (1989). The semantico-functional variability of words and the teaching of vocabulary to advanced EFL students. *Applied Linguistics*, 10(4), 382-391.

Kehe, D. & Kehe, P. (1996). Professors' expectations of foreign students in freshman-level courses. *JALT Journal*, 18 (1), 108-115.

Kennedy, G. (1991). "Between and through: The company they keep and the functions they serve." In: K. Aijmer & B. Altenberg (Eds.), *English corpus linguistics*, 95-110. New York: Longman Publishers.

Kernighan, B. W. & McIlroy, M. D. (Eds.) (1978). *UNIX programmer's manual (7th edition)*. Murray Hill, NJ: Bell Laboratories. (See [Hume&McIlroy 1990] for a newer edition.)

Kernighan, B. W. & Pike, R. (1984). *The UNIX programming environment*. Englewood Cliffs, NJ: Prentice Hall, Inc.

Kernighan, B. W. & Ritchie, D. M. (1988). *The C programming language (2nd edition)*. Englewood Cliffs, NJ: Prentice Hall, Inc.

Kroll, B. (Ed.) (1990a). *Second language writing*. Cambridge, UK: Cambridge University Press.

Kroll, B. (1990b). What does time buy? ESL student performance on home versus class compositions. In: B. Kroll (Ed.), *Second language writing*, 140-154. Cambridge, UK: Cambridge University Press.

Lee, K.-H. (1991). P-KIMMO: A Prolog implementation of the two level model.

ERIC Document Reproduction Service (EDRS), Doc. No. ED370437. `http://edrs.com/`

LEECH, G. & FLIGELSTONE, S. (1992). Computers and corpus analysis. IN: C. S. Butler (Ed.), *Computers and written texts*, Oxford: Basil Blackwell, Inc.

LEECH, G. & SVARTVIK, J. (1994). *A communicative grammar of English (2nd edition)*. New York: Longman Publishers.

LEKI, I. (1990). Coaching from the margins: Issues in written response. IN: B. Kroll (Ed.), *Second language writing*, 57-68. Cambridge, UK: Cambridge University Press.

LESK, M. E. & SCHMIDT, E. (1978). Lex — A lexical analyzer generator. IN: B. W. Kernighan & M. D. McIlroy, (Eds.), *UNIX programmer's manual (7th edition)*. Murray Hill, NJ: Bell Laboratories.
On-line `troff` file: `http://cm.bell-labs.com/7thEdMan/vol2/lex`

LEWIS, M. (1993). *The lexical approach: The state of ELT and a way forward*. LTP Teacher Training Series. Hove, UK: Language Teaching Publications.

MCDONALD, N. H., FRASE, L. T., GINGRICH, P. & KEENAN, S. (1982). The writer's workbench: Computer aids for text analysis. *Educational Psychologist*, 17, 172-179.

MEYER, C. F. & TENNEY, R. L. (1993). Tagger: An interactive tagging program. IN: C. Souter & E. Atwell (Eds.), *Corpus-based computational linguistics*. Amsterdam: Editions Rodopi, B. V.

MURAKAWA, H. (1995). Grammar dictionary and computer assisted composition. IN: J. Izzo (Ed.), *1994 Annual Review of The Center for Language Research*. The University of Aizu, Aizu-Wakamatsu, Japan. ERIC Document Reproduction Service (EDRS), Doc. No. ED394284. `http://edrs.com/`
`http://www.u-aizu.ac.jp/~izzo/Subconj94/subconj94.html`

MURRAY, D. E. (1996). Technology is driving the future, the steering is up to us. *TESOL Matters*, 6(3), June/July, p. 3.

NUNAN, D. & LAMB, C. (1996). *The self-directed teacher: Managing the learning process*. Cambridge, UK: Cambridge University Press.

OMAGGIO, A. C. (1986). *Teaching language in context*. Boston: Heinle & Heinle Publishers.

Orr, T., Christianson, K. T., DeHart, J., Izzo, J., Lambacher, S., Moore, C. H., Moore, W., Murakawa, H. & Washburn, N. (1995). A list of words and phrases related to computer science. Unpublished manuscript. The Center for Language Research, The University of Aizu, Aizu-Wakamatsu, Japan.

Owen, C. (1993). Corpus-based grammar and the Heineken effect: Lexico-grammatical descriptions for language learners. *Applied Linguistics*, 14(2), 167-187.

Quirk, R., Greenbaum, S., Leech, G. & Svartvik, J. (1985). *A comprehensive grammar of the English language*. New York: Longman Publishers.

Reid, J. (1990). Responding to different topic types: A quantitative analysis from a contrastive rhetoric perspective. In: B. Kroll (Ed.), *Second language writing*, 191-210. Cambridge, UK: Cambridge University Press.

Renouf, A. & Sinclair, J. M. (1991). Collocational frameworks in English. In: K. Aijmer & B. Altenberg (Eds.) *English corpus linguistics*, 128-143. New York: Longman Publishers.

Saito, Y. (1992). Western reading strategies and Japanese texts: The effects of punctuation deletion. Paper presented at the 42nd Annual Meeting of the National Reading Conference, San Antonio, Texas, USA (December 2-5). ERIC Document Reproduction Service (EDRS), Doc. No. ED353553. `http://edrs.com/`

Schreiner, A. T. & Friedman, H. G. (1985). *Introduction to compiler construction with UNIX*. Englewood Cliffs, NJ: Prentice Hall, Inc.

Scott, M. & Murison-Bowie, S. (1993). *MicroConcord manual: An introduction to the practices and principles of concordancing in language teaching*. Oxford: Oxford University Press.

Silberschatz, A., Korth, H. & Sudarshan, S. (1996). *Database system concepts*. New York: The McGraw-Hill Companies, Inc.

Souter, C. & Atwell, A. (Eds.) (1993). *Corpus-based computational linguistics.* Amsterdam: Editions Rodopi, B. V.

Tribble, C. & Jones, G. (1990). *Concordances in the classroom.* New York: Longman Publishers.

Wall, L. & Schwarz, R. L. (1990). *Programming perl.* Sebastopol, CA: O'Reilly & Associates.

Webb, J. H. M. (1992). 121 *common mistakes of Japanese students of English (Revised edition).* Tokyo: The Japan Times.

Willis, D. (1990). *The lexical syllabus, a new approach to language teaching.* London: Collins ELT.

Wolfram, S. (1991). *Mathematica — A system for doing mathematics by computer (2nd edition).* Reading, MA: Addison-Wesley Publishing Company.

Yang, H. (1994). Roles of the middleschool teacher in Japan & the United States. *Journal of Curriculum and Supervision,* 10 (1), 77-93.

BEST COPY AVAILABLE